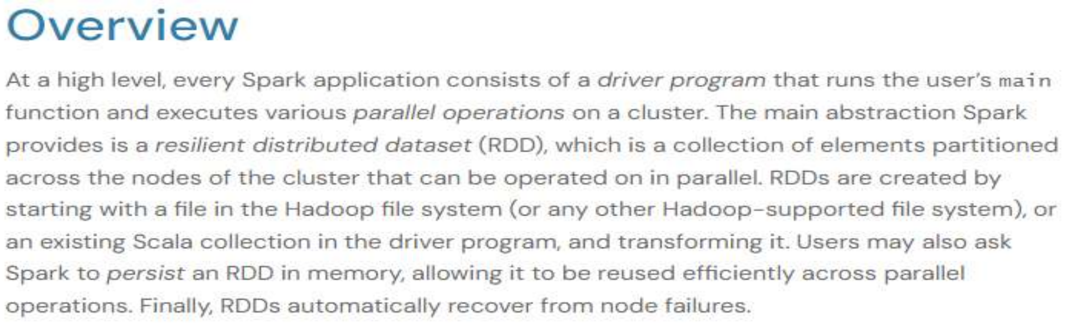
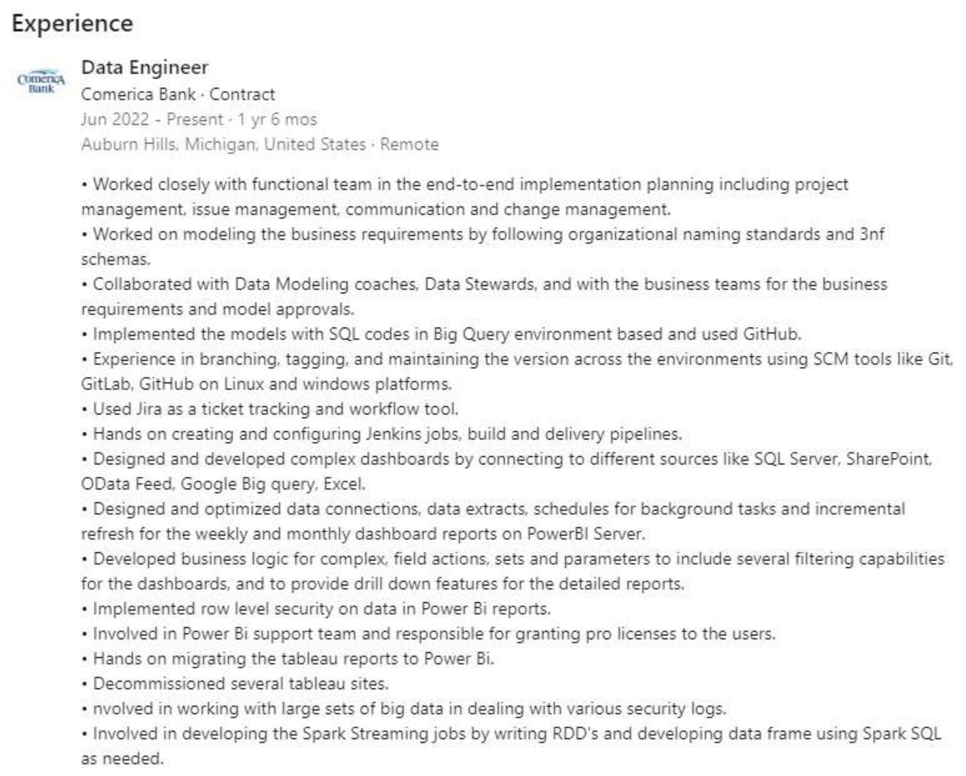
Exhibit 8

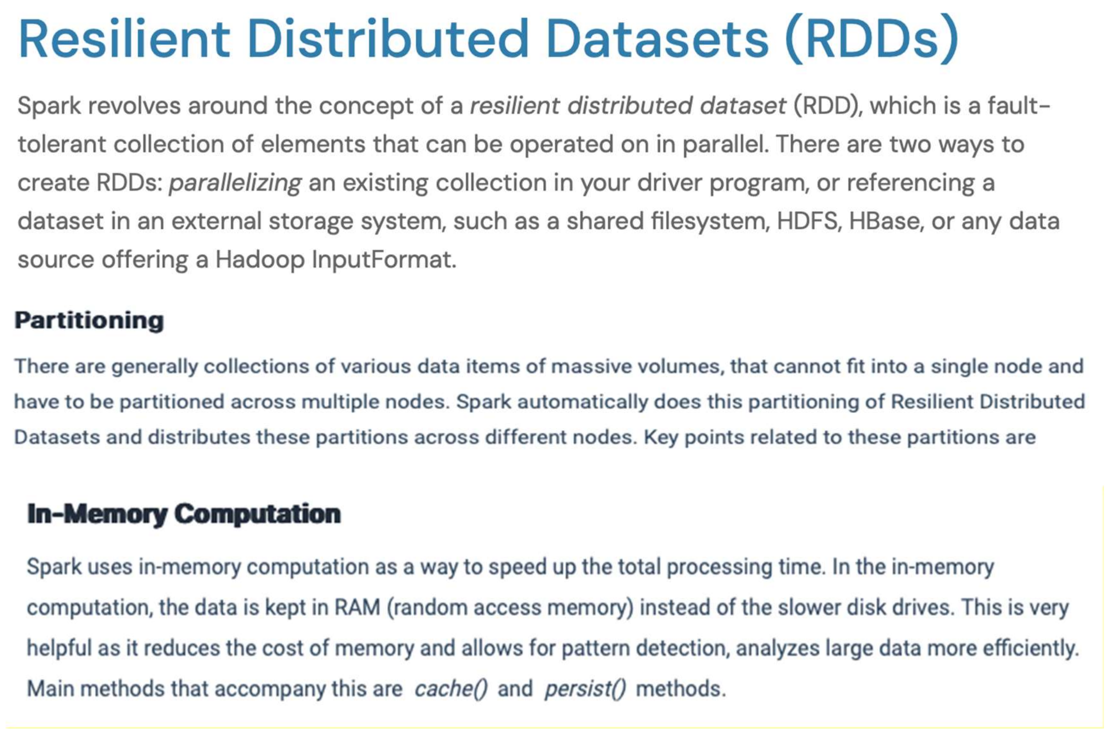
Exhibit 8 to Complaint
Intellectual Ventures I LLC and Intellectual Ventures II LLC

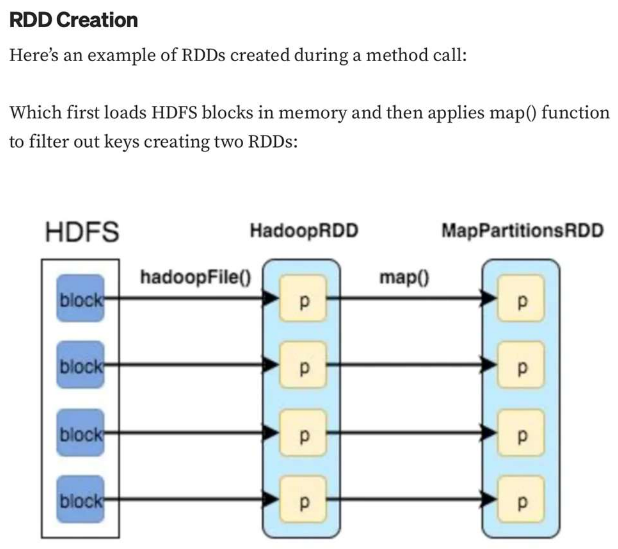
Example Comerica Count III Systems and Services
U.S. Patent No. 7,712,080 (“the ’080 Patent”)

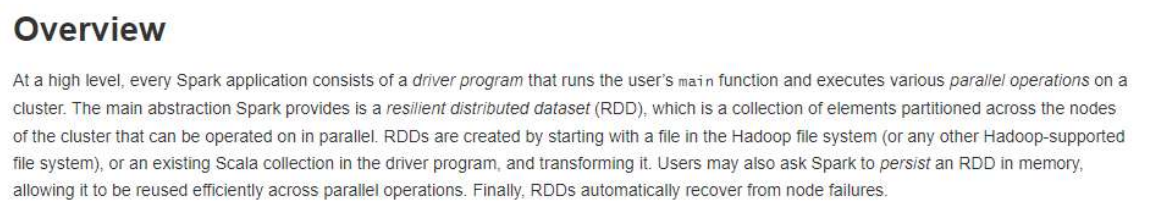
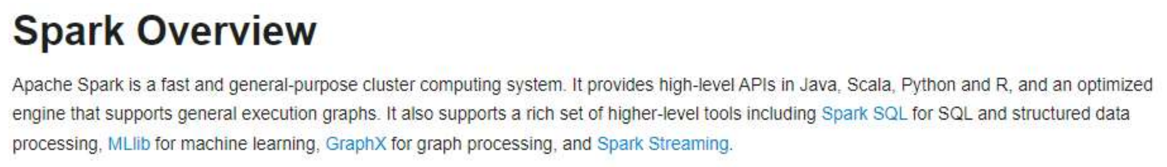
The Accused Instrumentalities include, without limitation, Comerica’s systems that utilize Apache Spark (“Spark”); all past, current, and future systems, and services that operate in the same or substantially similar manner as the specifically identified systems and services; and all past, current, and future Comerica’s systems, and services that have the same or substantially similar features as the specifically identified systems and services (“Example Comerica Count III Systems and Services”).

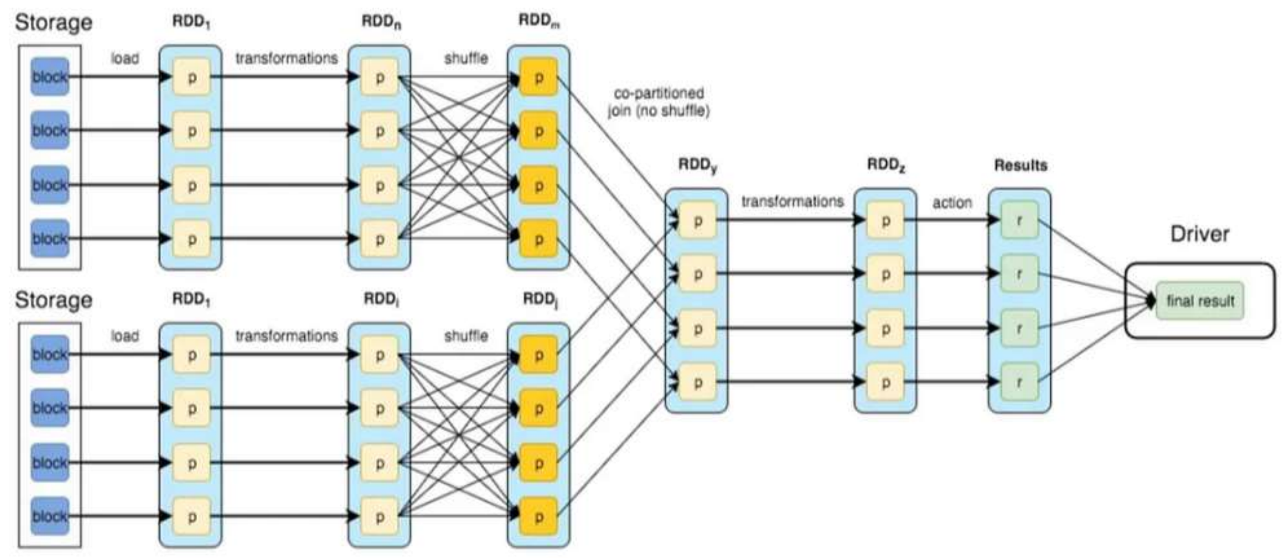
U.S. Patent No. 7,712,080	
Example Claim 9	**Example Comerica Count III Systems and Services**
9. A distributed parallel computing system, having at least one memory area and at least one processor, comprising:	Upon information and belief, Comerica's systems include "[a] distributed parallel computing system, ha ving at least one memory area and at least one processor[.]" On information and belief, Comerica's systems consist of a distributed parallel computing system having many memory areas and multiple processors. Upon information and belief, Comerica's distributed parallel computing system is implemented, at least in part, using Spark.  Overview At a high level, every Spark application consists of a *driver program* that runs the user's `main` function and executes various *parallel operations* on a cluster. The main abstraction Spark provides is a *resilient distributed dataset* (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to *persist* an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures. *See* https://spark.apache.org/docs/latest/rdd-programming-guide.html (last accessed on November 10, 2023).

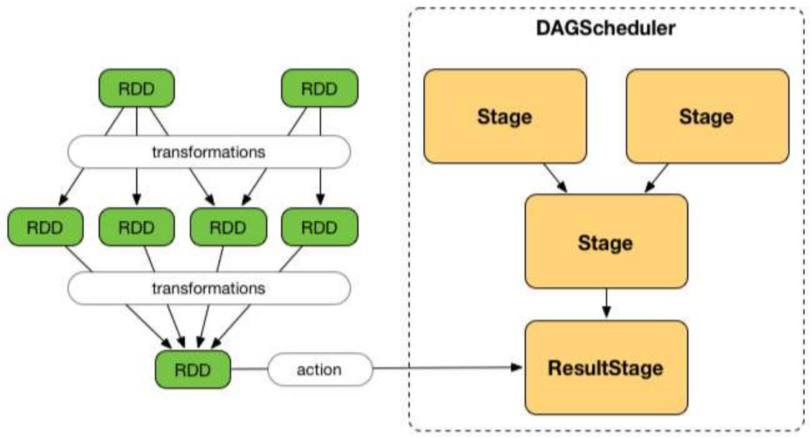
	 Experience Data Engineer Comerica Bank · Contract Jun 2022 - Present · 1 yr 6 mos Auburn Hills, Michigan, United States · Remote • Worked closely with functional team in the end-to-end implementation planning including project management, issue management, communication and change management. • Worked on modeling the business requirements by following organizational naming standards and 3nf schemas. • Collaborated with Data Modeling coaches, Data Stewards, and with the business teams for the business requirements and model approvals. • Implemented the models with SQL codes in Big Query environment based and used GitHub. • Experience in branching, tagging, and maintaining the version across the environments using SCM tools like Git, GitLab, GitHub on Linux and windows platforms. • Used Jira as a ticket tracking and workflow tool. • Hands on creating and configuring Jenkins jobs, build and delivery pipelines. • Designed and developed complex dashboards by connecting to different sources like SQL Server, SharePoint, OData Feed, Google Big query, Excel. • Designed and optimized data connections, data extracts, schedules for background tasks and incremental refresh for the weekly and monthly dashboard reports on PowerBI Server. • Developed business logic for complex, field actions, sets and parameters to include several filtering capabilities for the dashboards, and to provide drill down features for the detailed reports. • Implemented row level security on data in Power Bi reports. • Involved in Power Bi support team and responsible for granting pro licenses to the users. • Hands on migrating the tableau reports to Power Bi. • Decommissioned several tableau sites. • nvolved in working with large sets of big data in dealing with various security logs. • Involved in developing the Spark Streaming jobs by writing RDD's and developing data frame using Spark SQL as needed. *See* https://www.linkedin.com/in/shubodair/ (last accessed on November 10, 2023).

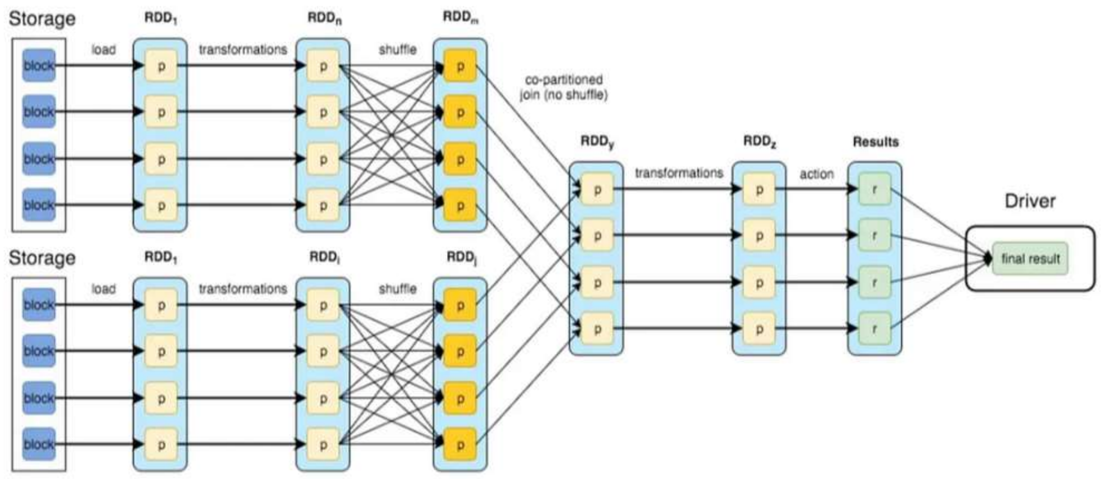
9[a] at least one distributed shared variable capable of loading into the at least one memory area, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically loaded into the at least one memory area; and	Upon information and belief, Comerica implements Spark. Upon information and belief, Comerica's systems include "at least one distributed shared variable capable of loading into the at least one memory area, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically loaded into the at least one memory area[.]" Spark collects large volumes of data that cannot fit on a single node and distributes slices of the data across multiple nodes. Spark automatically partitions the data into Resilient Distributed Datasets that operate in parallel. The Resilient distributed datasets are partitioned across multiple nodes and processed across at least one memory area.  Resilient Distributed Datasets (RDDs) Spark revolves around the concept of a *resilient distributed dataset* (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs: *parallelizing* an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat. **Partitioning** There are generally collections of various data items of massive volumes, that cannot fit into a single node and have to be partitioned across multiple nodes. Spark automatically does this partitioning of Resilient Distributed Datasets and distributes these partitions across different nodes. Key points related to these partitions are **In-Memory Computation** Spark uses in-memory computation as a way to speed up the total processing time. In the in-memory computation, the data is kept in RAM (random access memory) instead of the slower disk drives. This is very helpful as it reduces the cost of memory and allows for pattern detection, analyzes large data more efficiently. Main methods that accompany this are *cache()* and *persist()* methods. *See* https://spark.apache.org/docs/latest/rdd-programming-guide.html (last accessed on November 10, 2023).

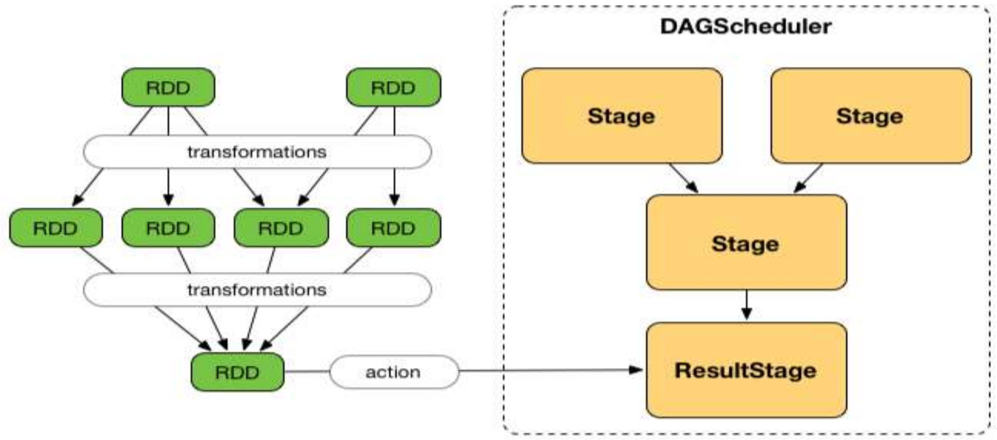
	The RDDs contain variables that allow the nodes to process slides of data in parallel. RDDs are processed using functions and multiple variables that can be used to create other RDDs to further process the data set.  **RDD Creation** Here's an example of RDDs created during a method call: Which first loads HDFS blocks in memory and then applies map() function to filter out keys creating two RDDs: HDFS HadoopRDD MapPartitionsRDD hadoopFile() map() block p *See* https://spark.apache.org/docs/latest/rdd-programming-guide.html (last accessed on November 10, 2023).

9[b][i] at least one distributed sequential computing program, configured to operate in the at least one processor, configured to access the at least one distributed shared variable,	Upon information and belief, Comerica's systems include "at least one distributed sequential computing program, configured to operate in the at least one processor, configured to access the at least one distributed shared variable[.]" Spark contains a driver program that is configured to operate in the at least one processor and is configured to access the RDDs. Spark jobs are authored in language such as Java, Scala, Python, or R. Spark is configured to access the at least one distributed shared variable (the RDDs) through the access of RDDs by a Spark job.  **Overview** At a high level, every Spark application consists of a *driver program* that runs the user's `main` function and executes various *parallel operations* on a cluster. The main abstraction Spark provides is a *resilient distributed dataset* (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to *persist* an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures. *See* https://spark.apache.org/docs/1.2.1/programming-guide.html#:~:text=Go%20from%20Here-,Overview,parallel%20operations%20on%20a%20cluster. (last accessed November 10, 2023).  **Spark Overview** Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming. *See* https://spark.apache.org/docs/2.0.1/ (last accessed on November 10, 2023).

9[b][ii] and configured to transform into at least one distributed parallel computing program by spawning at least one child distributed sequential computing system program	Upon information and belief, Comerica's systems are "configured to transform into at least one distributed parallel computing program by spawning at least one child distributed sequential computing system program[.]" The Spark Execution Model transforms the distributed sequential computing program into a series of stages and sets of parallel tasks that run in parallel on the Spark cluster. This includes Spark tasks operating on a set of RDDs in parallel in a given stage.  Storage RDD1 RDDn RDDm load transformations shuffle block p co-partitioned join (no shuffle) RDDy RDDz Results transformations action r Driver final result Storage RDD1 RDDi RDDj load transformations shuffle *See* https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167 (last accessed on November 10, 2023).

9[b][iii] when at least one intermediate condition occurs within the at least one distributed sequential program	Upon information and belief, Comerica's systems include "at least one intermediate condition occurs within the at least one distributed sequential program[.]" Spark implements shuffle boundary where stages/tasks must wait for the previous stage to finish before they fetch map outputs. DAGScheduler splits up a job into a collection of stages. Each stage contains a sequence of narrow transformations that can be completed without shuffling the entire data set, separated at **shuffle boundaries**, i.e. where shuffle occurs. Stages are thus a result of breaking the RDD graph at shuffle boundaries.  DAGScheduler RDD RDD transformations RDD RDD RDD RDD transformations RDD action Stage Stage Stage ResultStage Figure 4. Graph of Stages Shuffle boundaries introduce a barrier where stages/tasks must wait for the previous stage to finish before they fetch map outputs. *See* https://mallikarjuna_g.gitbooks.io/spark/content/spark-dagscheduler-stages.html (last accessed on November 10, 2023).

9[b][iv] wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations,	Upon information and belief, Comerica's systems are configured such that "wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations[.]" Spark implements a set of tasks running in parallel for a given stage. The distributed parallel computing program includes the Spark job and driver program (above). The spawned child distributed sequential computing program is the sequential code in a task for a given stage (above).  Storage RDD1 RDDn RDDm load transformations shuffle block p co-partitioned join (no shuffle) RDDy RDDz Results transformations action r Driver final result Storage RDD1 RDDi RDDj load transformations shuffle *See* https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167 (last accessed on November 10, 2023).

9[b][v] wherein the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation.	Upon information and belief, Comerica's systems are configured such that "wherein the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation." Spark implements shuffle boundary where stages/tasks must wait for the previous stage to finish before they fetch map outputs. DAGScheduler splits up a job into a collection of stages. Each stage contains a sequence of narrow transformations that can be completed without shuffling the entire data set, separated at **shuffle boundaries**, i.e. where shuffle occurs. Stages are thus a result of breaking the RDD graph at shuffle boundaries.  DAGScheduler RDD RDD transformations RDD RDD RDD RDD transformations RDD action Stage Stage Stage ResultStage Figure 4. Graph of Stages Shuffle boundaries introduce a barrier where stages/tasks must wait for the previous stage to finish before they fetch map outputs. *See* https://mallikarjuna_g.gitbooks.io/spark/content/spark-dagscheduler-stages.html (last accessed on November 10, 2023).